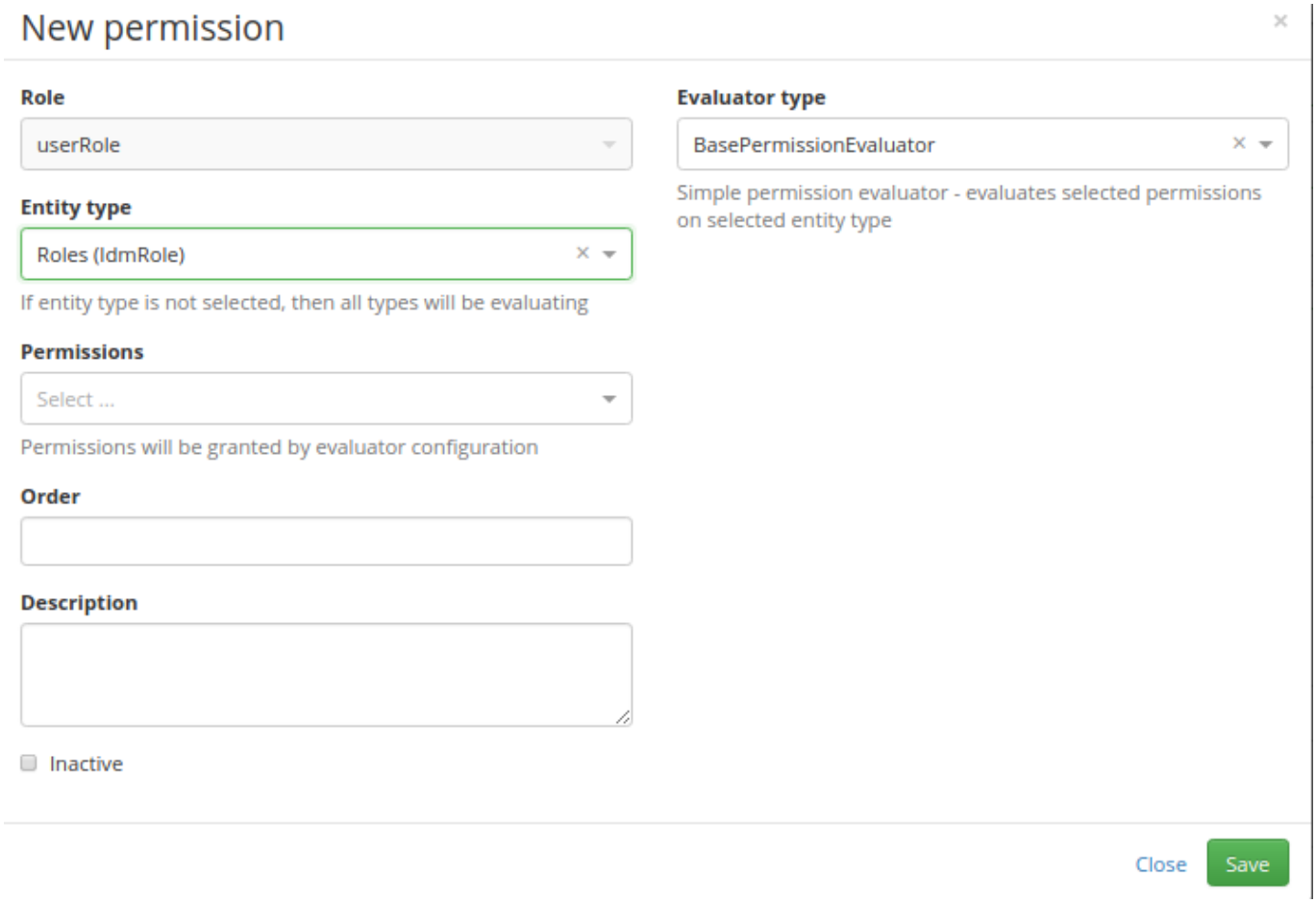
Authorization - Create a new evaluator

Step 1. create evaluator

The most *easiest* way is to copy some existing evaluator and just refactor its name, package, constructor and everything you need. Good for copy is for example RoleGuaranteeEvaluator, this evaluator returns all roles, of which the logged user is the guarantee.

As you can see RoleGuaranteeEvaluator extends AbstractAuthorizationEvaluator and as a generic type there is the class IdmRole (generic type is always entity **not DTO!**) On the next picture you can see that generic type is on FE shown as **Entity type**.



Step 2. two main methods

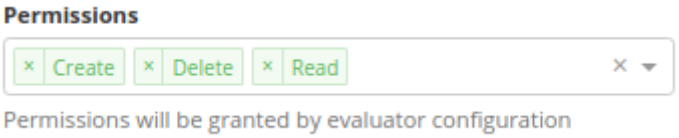
In almost every evaluator is two main methods in which is written almost complete logic. These methods is **getPredicate** and **getPermissions**. Simple description these methods (if you want some betyter description go to Authorization policies):

getPredicate

getPredicate is method in which we will **add** access to data. Evaluators only add, never limit the given set. Example: if you have some evaluator that add all identities you will be not able to remove some identities or annul this set. You are able only to extends set.

getPermissions

In this method you can add or remove **Permissions**. From super class you get empty set. super.getPermissions(entity, policy); If you want add all permission that it was set via frontend use this (Object AuthorizationPolicy contains all permission that is set via FE): permissions.addAll(policy.getPermissions()); Basic permission: CRUD (Create, Read, Udpdate, Delete), Administration (all of CRUD), View in select box (autocomplete). Example of setting permissions via FE:



Add some logic

Now we have prepare basic evaluator that not contains logic.

```
@Component
@Description("Testing evaluator.")
public class TestingEvaluator extends
AbstractAuthorizationEvaluator<IdmRole> {


    @Autowired
    public TestingEvaluator() {
    }

    @Override
    public Predicate getPredicate(Root<IdmRole> root, CriteriaQuery<?>
query, CriteriaBuilder builder, AuthorizationPolicy policy,
BasePermission... permission) {
        return null;
    }

    @Override
    public Set<String> getPermissions(IdmRole entity, AuthorizationPolicy
policy) {
        Set<String> permissions = super.getPermissions(entity, policy);
        permissions.addAll(policy.getPermissions());
        return permissions;
    }
}
```

Our evaluator will be add access for user to roles that name is ended with some value from application properties. At first we will need autowire these beans: **SecurityService** (for actual logged user) and **ConfigurationService** (for read application properties), all these beans are from package **eu.bcvsolutions.idm.core.**.

Constructor after autowire:

```
private final SecurityService securityService;
private final ConfigurationService configurationService;

@Autowired
public TestingEvaluator(
        SecurityService securityService,
        ConfigurationService configurationService) {
    //
    Assert.notNull(securityService);
    Assert.notNull(configurationService);
    //
    this.securityService = securityService;
    this.configurationService = configurationService;
}
```

To method **getPredicate** we will now add check for current logged user.

```
AbstractAuthentication authentication = securityService.getAuthentication();
if (authentication == null || authentication.getCurrentIdentity() == null) {
    return null;
}
```

Now we are add get value from application properties and build query:

```
String valueFromProperties =
configurationService.getValue("some_attribute");
//
if (hasPermission(policy, permission)) {
    return query.where(
            builder.like(
                    root.get(IdmRole_.name), "%" + valueFromProperties
                    )
            ).getRestriction();
}
return null;
```

As you see from configurationService we get attribute with key *some_attribute* and after that we build simple query.

Finished evaluator

There is finished evaluator:

```
@Component
@Description("Testing evaluator.")
public class TestingEvaluator extends
AbstractAuthorizationEvaluator<IdmRole> {

    private final SecurityService securityService;
    private final ConfigurationService configurationService;

    @Autowired
    public TestingEvaluator(SecurityService securityService,
ConfigurationService configurationService) {
        //
        Assert.notNull(securityService);
        Assert.notNull(configurationService);
        //
        this.securityService = securityService;
        this.configurationService = configurationService;
    }

    @Override
    public Predicate getPredicate(Root<IdmRole> root, CriteriaQuery<?>
query, CriteriaBuilder builder,
            AuthorizationPolicy policy, BasePermission... permission) {
        AbstractAuthentication authentication =
securityService.getAuthentication();
        if (authentication == null || authentication.getCurrentIdentity() ==
null) {
            return null;
        }
        //
        String valueFromProperties =
configurationService.getValue("some_attribute");
        //
        if (hasPermission(policy, permission) &&
!StringUtils.isEmpty(valueFromProperties)) {
            return query.where(builder.like(root.get(IdmRole_.name),
valueFromProperties)).getRestriction();
        }
        return null;
    }

    @Override
    public Set<String> getPermissions(IdmRole entity, AuthorizationPolicy
policy) {
        Set<String> permissions = super.getPermissions(entity, policy);
        //
        String valueFromProperties =
configurationService.getValue("some_attribute");
        //
        if (!StringUtils.isEmpty(valueFromProperties) &&
```

```
entity.getName().endsWith(valueFromProperties)) {
            permissions.addAll(policy.getPermissions());
        }
        return permissions;
    }
}
```